Lab Exercise 6– Terraform Multiple tfvars Files

Objective:

Learn how to use multiple tfvars files in Terraform for different environments.

Prerequisites:

•Terraform installed on your machine.
•Basic knowledge of Terraform configuration and variables.

Steps:
1. Create a Terraform Directory:

```
rohin@victus:~/UPES/Sem_6/SPCM/Experiment 6$ mkdir terraform-multiple-tfvars
cd terraform-multiple-tfvars
rohin@victus:~/UPES/Sem_6/SPCM/Experiment 6/terraform-multiple-tfvars$
```

•Create Terraform Configuration Files:
•Create a file named main.tf:

#main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "5.35.0"
    }
  }
}


provider "aws" {
region = var.region
access_key = "AKIAWMVRZHDWFWGTVYN6"
secret_key = "+Vl0tT/33xY0hKkIxeCo0feZdouS362xLT9O94SF"

}
resource "aws_instance" "example" {
ami= var.ami
instance_type = var.instance_type
}
```

•Create a file named variables.tf:

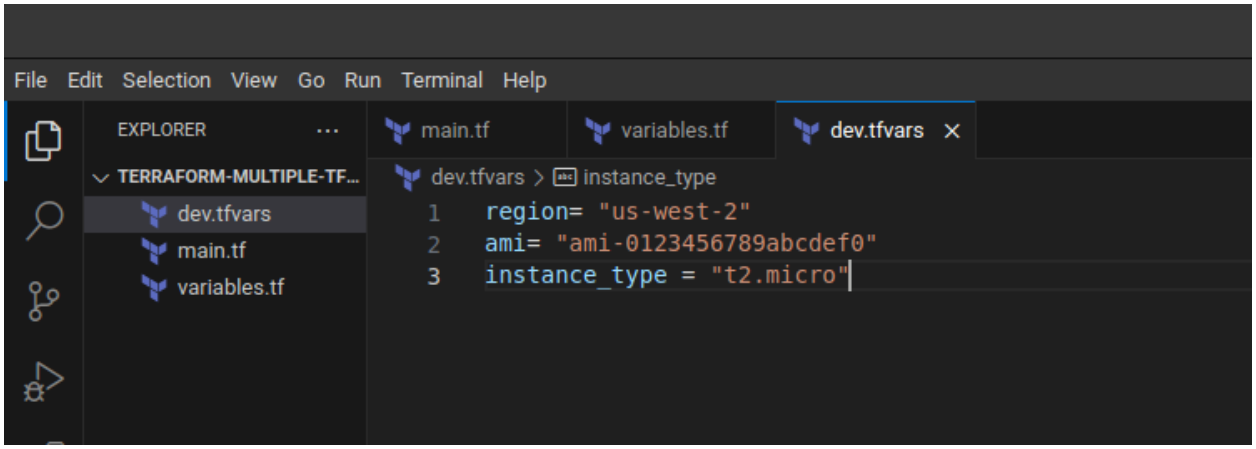
variables.tf

```
variable "region" {
description = "AWS region"
default = "us-west-2"
}
variable "ami" {
description = "AMI ID"
default = "ami-0c55b159cbfafe1f0"
}
variable "instance_type" {
description = "EC2 Instance Type"
default = "t2.micro"
}
```

2. Create Multiple tfvars Files:

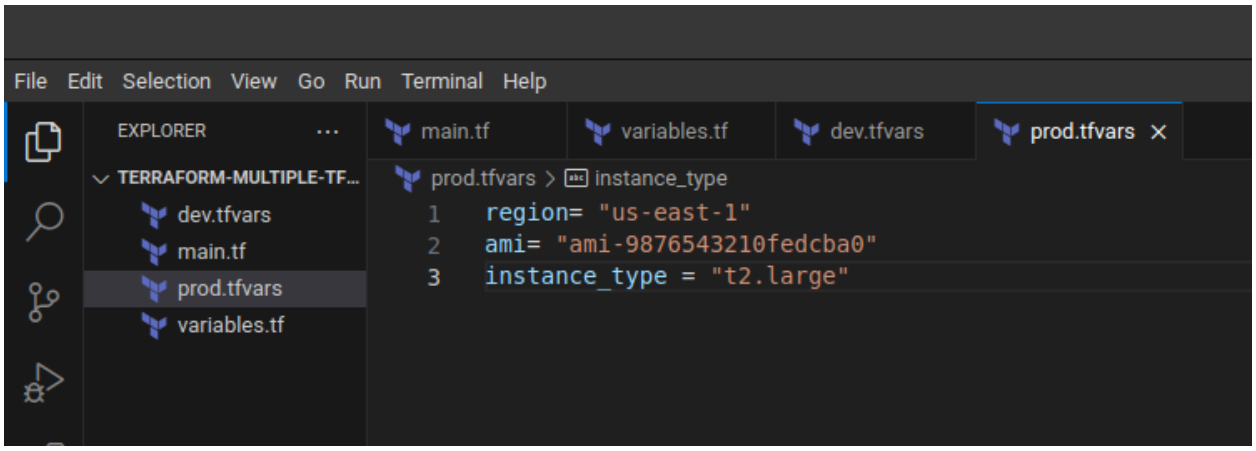
•Create a file named dev.tfvars:

dev.tfvars



•Create a file named prod.tfvars:

prod.tfvars



•In these files, provide values for the variables based on the environments.

3. Initialize and Apply for Dev Environment:

•Run the following Terraform commands to initialize and apply the configuration for the dev environment:

```
rohin@victus: ~/UPES/Sem_6/SPCM/Experiment 6/terraform-multiple-tfvars
rohin@victus:~/UPES/Sem_6/SPCM/Experiment 6/terraform-multiple-tfvars$ terraform init
terraform apply -var-file=dev.tfvars

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.35.0"...
- Installing hashicorp/aws v5.35.0...
- Installed hashicorp/aws v5.35.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create
```

4. Initialize and Apply for Prod Environment:

•Run the following Terraform commands to initialize and apply the configuration for the prod environment:

```
rohin@victus: ~/UPES/Sem_6/SPCM/Experiment 6/terraform-multiple-tfvars
rohin@victus:~/UPES/Sem_6/SPCM/Experiment 6/terraform-multiple-tfvars$ terraform init
terraform apply -var-file=prod.tfvars

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.35.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.example will be created
  + resource "aws_instance" "example" {
      + ami                                  = "ami-9876543210fedcba0"
      + arn                                  = (known after apply)
```

5. Test and Verify:

•Observe how different tfvars files are used to set variable values for different environments during the apply process.

•Access the AWS Management Console or use the AWS CLI to verify the creation of resources in the specified regions and instance types.

6. Clean Up:

•After testing, you can clean up resources:

```
terraform destroy -var-file=dev.tfvars
terraform destroy -var-file=prod.tfvars
```

•Confirm the destruction by typing yes.